WHAT IS CLAIMED IS:

1. A double-ended concurrent shared object organized as a dynamically sized bi-directional referencing chain of nodes, the double-ended concurrent shared object employing distinguishing values to indicate spare nodes thereof and supporting concurrent non-interfering opposing-end accesses for states of two or more values.

2. The double-ended concurrent shared object of claim 1, wherein the concurrent non-interfering opposing-end accesses include pop-type accesses.

3. The double-ended concurrent shared object of claim 1, wherein the concurrent opposing-end accesses are push- and pop-type accesses, respectively, and wherein the push- and pop-type accesses are non-interfering for states of one or more values.

4. The double-ended concurrent shared object of claim 1, wherein the concurrent opposing-end accesses are push-type accesses, and wherein the push-type accesses are non-interfering for all states.

5. The double-ended concurrent shared object of claim 1, further supporting at least one spare node maintenance operation.

6. The double-ended concurrent shared object of claim 1, wherein the distinguishing values include opposing-end and terminal node variants thereof.

7. The double-ended concurrent shared object of claim 6, wherein the distinguishing values further include opposing-end terminal node variants.

8. The double-ended concurrent shared object of claim 6, wherein the distinguishing values further include at least one dead node marker variant.

9. The double-ended concurrent shared object of claim 1, embodied as a doubly-linked list of nodes allocated from a shared memory of a multiprocessor and access operations executable by processors thereof.

10. The double-ended concurrent shared object of claim 1, embodied as a computer program product encoded in media, the computer program product defining a data structure instantiable in shared memory of a multiprocessor and instructions executable thereby implementing access operations.

11. The double-ended concurrent shared object of claim 10,
wherein the data structure includes a double-ended queue; and
wherein the access operations include opposing-end variants of push and pop operations.

12. The double-ended concurrent shared object of claim 1, embodied as a doubly-linked list of nodes allocated from a memory of a processor and access operations executable thereby.

13. The double-ended concurrent shared object of claim 1, embodied as a computer program product encoded in media, the computer program product defining a data structure instantiable in memory of a processor and instructions executable thereby implementing access operations.

14. The double-ended concurrent shared object of claim 1,
wherein each of the nodes that are severed from the referencing chain are explicitly reclaimed by a respective process that destroys a last pointer thereto.

15. The double-ended concurrent shared object of claim 1,
wherein those of the nodes that are severed from the referencing chain are reclaimed by an automatic storage reclamation facility of an execution environment.

16. A method of facilitating concurrent programming using a dynamically-sized, linked-list representation of a double ended queue (deque), the method comprising:

encoding the deque using a subset of nodes of the linked-list, the linked-list including spare nodes at either or both ends of the deque;

defining opposing-end variants of push and pop access operations on the deque; and

defining opposing-end variants of at least one spare node maintenance operation,

wherein execution of any of the access and spare node maintenance operations is linearizable and non-blocking with respect to any other execution of the access and spare node maintenance operations.

17. The method of claim 16, further comprising:

employing left and right sentinel nodes of the linked-list to delimit the deque, wherein the left and right sentinel nodes and any spare nodes beyond a respective sentinel node encode a distinguishing value in a value field thereof.

18. The method of claim 17, further comprising:

employing opposing-end and terminal node variants of the distinguishing value.

19. The method of claim 18, further comprising:

employing opposing-end terminal node variants of the distinguishing value.

20. The method of claim 18, further comprising:

employing at least one dead node marker variant of the distinguishing value.

21. The method of claim 16,

wherein each of the access operations includes a synchronization operation targeting both a respective sentinel node and a value of a corresponding node, thereby ensuring linearizable and non-blocking execution with respect to any other execution of an access operation.

22. The method of claim 16,

wherein each of the spare node maintenance operations includes a synchronization operation targeting both a respective target node and a value of a corresponding node, thereby ensuring linearizable and non-blocking execution with respect to any other execution of an access or spare node maintenance operation.

23. The method of claim 16,
wherein each of the pop access operations includes a single synchronization operation per uncontended execution path thereof.

24. The method of claim 16,
wherein, if a suitable spare node is available, each of the push access operations includes a single synchronization operation per uncontended execution path thereof.

25. The method of claim 16,
wherein overhead associated with execution of each of the spare node maintenance operations is amortizable over multiple executions of the access operations that target a particular maintained node.

26. The method of claim 16,
wherein the at least one spare node maintenance operation is an add-type maintenance operation and includes a single synchronization operation per uncontended execution path thereof.

27. The method of claim 26,
wherein the at least one spare node maintenance operation further includes a remove-type maintenance operation that employs a dead node distinguishing value encoding to facilitate at least detection of a spur condition.

28. The method of claim 21,
wherein for at least some of the access operations, the synchronization operation is a Double Compare And Swap (DCAS) operation.

29. The method of claim 21,
wherein for at least some of the access operations, the synchronization operation is an N-way Compare And Swap (NCAS) operation.

30. The method of claim 21,
wherein for at least some of the access operations, the synchronization operation employs transactional memory.

31. The method of claim 16,
wherein the at least one spare node maintenance operation includes opposing-end variants of both add-type and remove-type operations.

32. A concurrent double ended queue (deque) representation encoded in one or more computer readable media, the deque representation comprising:
a doubly-linked list of nodes, including an interior subset thereof encoding the deque, left and right sentinel ones immediately adjacent to the interior subset, and one or more spare nodes beyond each of the left and right sentinel nodes;
push and pop access operations executable to access each of opposing ends of the deque; and
spare node maintenance operations executable to control numbers of the spare nodes beyond the left and right sentinel nodes,
wherein execution of any of the access and spare node maintenance operations is linearizable and non-blocking with respect to any other execution of the access and spare node maintenance operations.

33. The deque representation of claim 32, further comprising:
separate left sentinel and right sentinel identifier storage; and
separate value storage associated with each of the nodes of the list, wherein a distinguishing value encoded therein is distinguishable from a literal or pointer value,
wherein each of the access operations employs a synchronization operation to ensure linearizable modification of corresponding sentinel identifier

storage and value storage, despite concurrent execution of conflicting ones of the access operations.

34. The deque representation of claim 33, wherein the distinguishing value includes three variants thereof, respectively indicative of:

a terminal node;

a non-terminal left spare or sentinel node; and

a non-terminal right spare or sentinel node.

35. The deque representation of claim 33, wherein the distinguishing value includes four variants thereof, respectively indicative of:

a left terminal node;

a right terminal node;

a non-terminal left spare or sentinel node; and

a non-terminal right spare or sentinel node.

36. The deque representation of claim 33, wherein the distinguishing value includes at least five variants thereof, respectively indicative of:

a left terminal node;

a right terminal node;

a dead node;

a non-terminal left spare or sentinel node; and

a non-terminal right spare or sentinel node.

37. The deque representation of claim 33, wherein the synchronization operation employed by each one of the access operations is selected from the set of:

a Double Compare And Swap (DCAS) operation; and

an N-way Compare And Swap (NCAS) operation.

38. The deque representation of claim 33, wherein the synchronization operation employed by each one of the access operations employs transactional memory.

39. The deque representation of claim 33,

wherein the synchronization operation employed by each one of the access operations is not necessarily the same.

40. The deque representation of claim 32,
wherein the spare node maintenance operations include add-type spare node operations.

41. The deque representation of claim 32,
wherein the spare node maintenance operations include both add-type and remove-type spare node operations.

42. The deque representation of claim 33,
wherein the spare node maintenance operations include a remove-type spare node operation operable at a chop point; and
wherein left and right variants of the distinguishing value are themselves distinguishable.

43. The deque representation of claim 33,
wherein the spare node maintenance operations operate on the list at respective target nodes; and
wherein each of the spare node maintenance operations includes a synchronization operation to ensure linearizable modification of a pointer to the respective target node and corresponding value storage, despite concurrent execution of conflicting ones of the access and spare node maintenance operations.

44. The deque representation of claim 32,
wherein at least the nodes are allocated from a garbage-collected memory space.

45. A method of managing access to elements of a sequence encoded in a linked-list susceptible to concurrent accesses to one or both ends of the sequence, the method comprising:

encoding the sequence using a subset of nodes of the linked-list, the linked-list including spare nodes at at least one end of the subset of sequence encoding nodes;

mediating the concurrent accesses using a linearizable synchronization operation operable on an end-of-sequence identifier and a corresponding node value, wherein node values distinguish between sequence encoding nodes and spare nodes; and

in response to a depletion of the spare nodes, adding one or more additional nodes to the linked-list.

46. The method of claim 45, further comprising:

in response to an excess of the spare nodes, removing one or more of the spare nodes from the linked-list.

47. The method of claim 45,

wherein the node values further distinguish between terminal nodes and spare nodes.

48. The method of claim 45,

wherein the sequence is susceptible to access at both ends thereof, and

wherein the node values further distinguish spare nodes at one end from those at the other.

49. The method of claim 45,

wherein the sequence encoding nodes represent a double ended queue (deque);

wherein the concurrent accesses include add and remove operations at each end of the deque; and

wherein the adding of one or more additional nodes is performed at each end of the deque in response to a depletion of the spare nodes at that respective end of the deque.

50. The method of claim 45,

wherein the sequence encoding nodes represent a stack;

wherein the concurrent accesses include add and remove operations at a top-end of the stack; and

wherein the adding of one or more additional nodes is performed at the top-end in response to a depletion of the spare nodes at the top-end.

51. The method of claim 45,

wherein the sequence encoding nodes represent a queue;

wherein the concurrent accesses include add and remove operations at respective ends of the queue.

52. The method of claim 45, wherein the concurrent accesses include less than all of:

a first-end add operation;

a first-end remove operation;

a second-end add operation; and

a second-end remove operation.

53. The method of claim 45,

wherein a subset of the concurrent accesses are performed only by a single process or processor.

54. The method of claim 45,

wherein at least some instances of the linearizable synchronization operation include a double compare and swap (DCAS) operation.

55. The method of claim 45,

wherein at least some instances of the linearizable synchronization operation employ transactional memory.

56. A concurrent shared object representation encoded in one or more computer readable media, the concurrent shared object representation comprising:

a doubly-linked list of nodes, each having a left pointer, a right pointer and a value;

a pair of shared variables that identify respective left and right sentinel ones of the nodes, each encoding a distinguishing value;

a sequence of zero or more values encoded using respective ones, zero or more, of the nodes linked between the left and right sentinel nodes in the list;

spare ones of the nodes beyond either or both of the left and right sentinel nodes in the list;

access operations defined for access to opposing ends of the sequence; and

spare node maintenance operations defined to add additional spare nodes to and remove excess spare nodes from the list,

wherein concurrent operation of competing ones of the access and spare node maintenance operations is mediated by linearizable synchronization operations.

57. A computer program product encoded in at least one computer readable medium, the computer program product comprising:

functional sequences implementing left- and right-end access operations and at least one spare node maintenance operation on a double-ended concurrent shared object instantiable as a doubly-linked list of nodes, including an interior subset thereof encoding a double-ended sequence, left and right sentinel nodes immediately adjacent the interior subset, and one or more spare nodes beyond each of the left and right sentinel nodes,

wherein instances of the functional sequences are concurrently executable by plural execution units and each include a linearizable synchronization operation to mediate competing executions of the functional sequences.

58. The computer program product of claim 57, wherein the spare node maintenance operations include:

both add- and remove-type operations.

59. The computer program product of claim 57, wherein the access operations include:

the left- and right-end remove-type operations; and

at least one insert-type operation.

60. The computer program product of claim 57, wherein the access operations include:

the left- and right-end insert-type operations; and

at least one remove-type operation.

61. The computer program product of claim 57, wherein the access operations include left- and right-end push and pop operations.

62. The computer program product of 57,

wherein the at least one computer readable medium is selected from the set of

a disk, tape or other magnetic, optical, or electronic storage medium and a network, wireline, wireless or other communications medium.

63. An apparatus comprising:

plural processors;

one or more stores addressable by the plural processors;

left and right identifiers accessible to each of the plural processors for

identifying a double-ended sequence represented by an interior subset of nodes of a doubly-linked list encoded in the one or more stores, the doubly-linked list including left and right sentinel nodes immediately adjacent the interior subset and one or more spare nodes beyond each of the left and right sentinel nodes; and

means for coordinating competing left- and right-end access operations and at least one spare node maintenance operation on the list, the

coordinating means employing instances of a linearizable synchronization operation and distinguishing node value encodings.

64. The apparatus of claim 63,

means for explicitly reclaiming a node severed from the list.